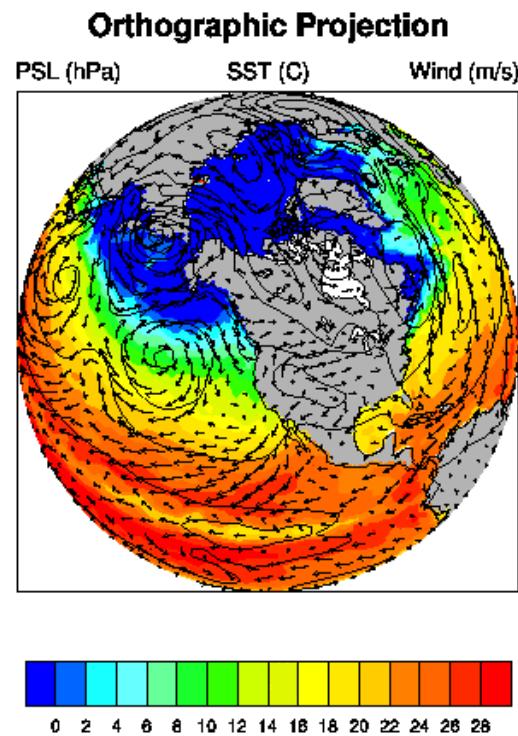
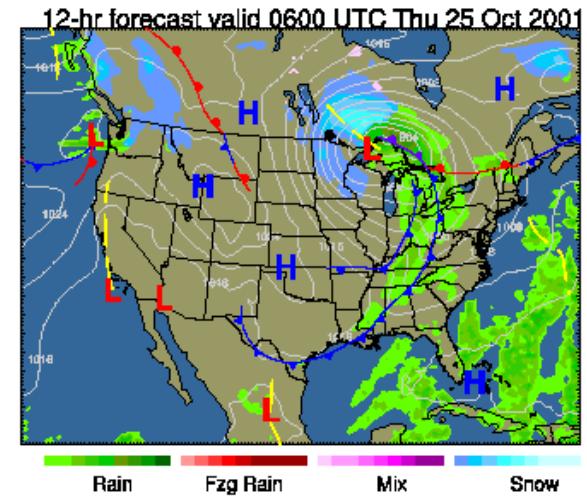
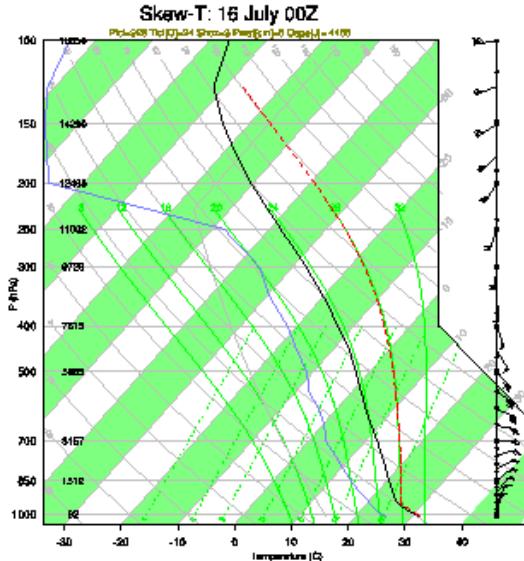


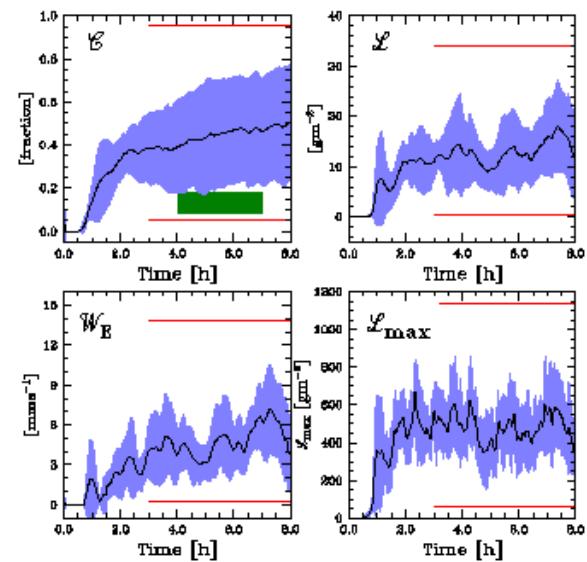
# NCL Workshop Overview

Dennis Shea

National Center for  
Atmospheric Research



## Simulations of Tradewind Cumuli Ensemble Means



# Tutorial Overview

## Objective

- comfortable with NCL, NCO and CMPS
- access, process and visualize data

## NCAR Command Language [NCL]

- **day 1:**
  - netCDF model, Fortran/C compare, language syntax basics
  - File I/O
  - Graphics
- **day 2:**
  - Data Processing (2 morning sessions)

## Command line operators [optional]

- **day 3:**
  - netCDF operators (NCO)
  - Climate Model Processing Suite (CMPS)

# NCL-SCD/CCSM-CGD History

- **CGD developed CCM Processor**

- development began in early 70s
- processed atmospheric model output (very model-centric)
- monolithic fortran code with many Cray features
- portability problems; poor graphics; CCMHT only

- **wanted tool with broader capabilities**

- portable, supported
- multiple data formats; quality graphics; processing capabilities; use external fortran/C codes

- **competition: IDL, matlab, yorick, NCL**

- required to perform 10 specific tasks (based on CCM)
- vote taken ... (1) NCL (2) yorick .... (3) IDL (4) matlab
- desire for free tool was big factor
- facilitate sharing of data processing scripts

# CGD-SCD commitment to help !

- **CGD-CSM-SCD Support**

- WWW tutorials
  - numerous downloadable examples to get you going
- downloadable reference manuals, FAQ
- workshops
- **most important: people to help you**

**murphys@ucar.edu**

[cc: haley@ucar.edu , shea@ucar.edu]

**ncl-talk:** <http://ngwww.ucar.edu/ncl-talk/>

- **Class will not make you an NCL expert**
  - minimize ‘angst’ associated with new language
- **You MUST invest time to learn**
  - “Osmosis” method of learning does not work
  - NCL learning curve similar to IDL/Matlab/yorick

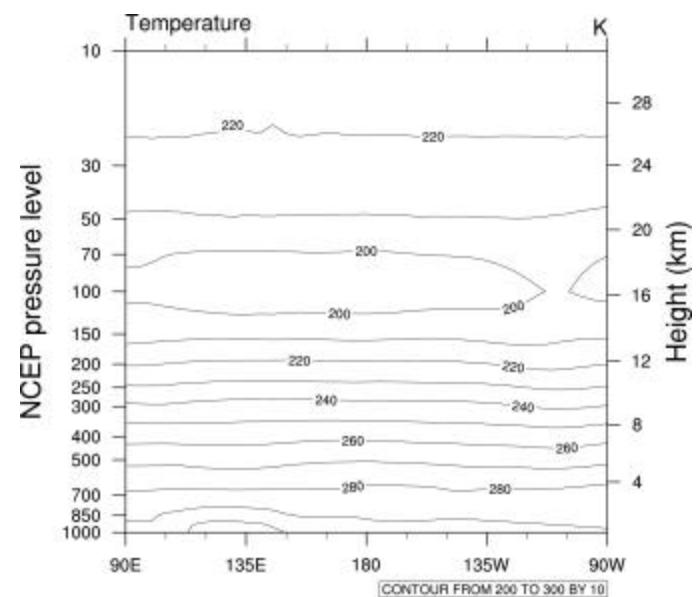
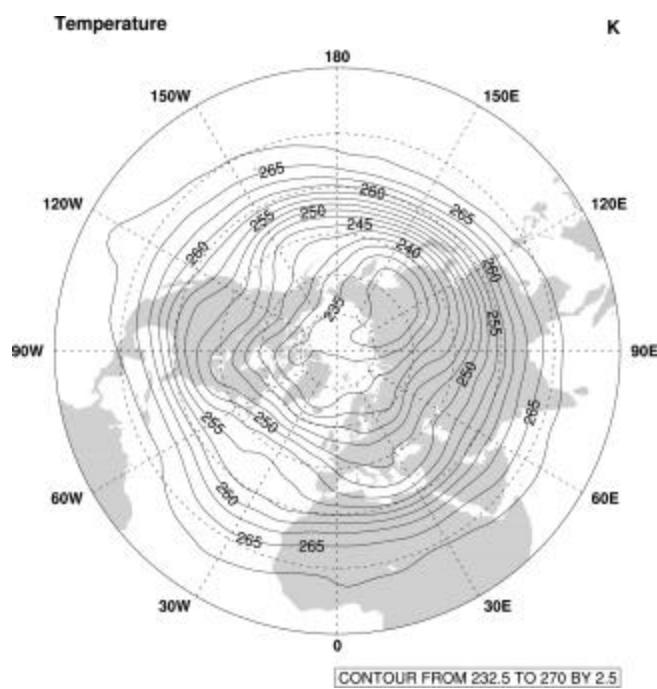
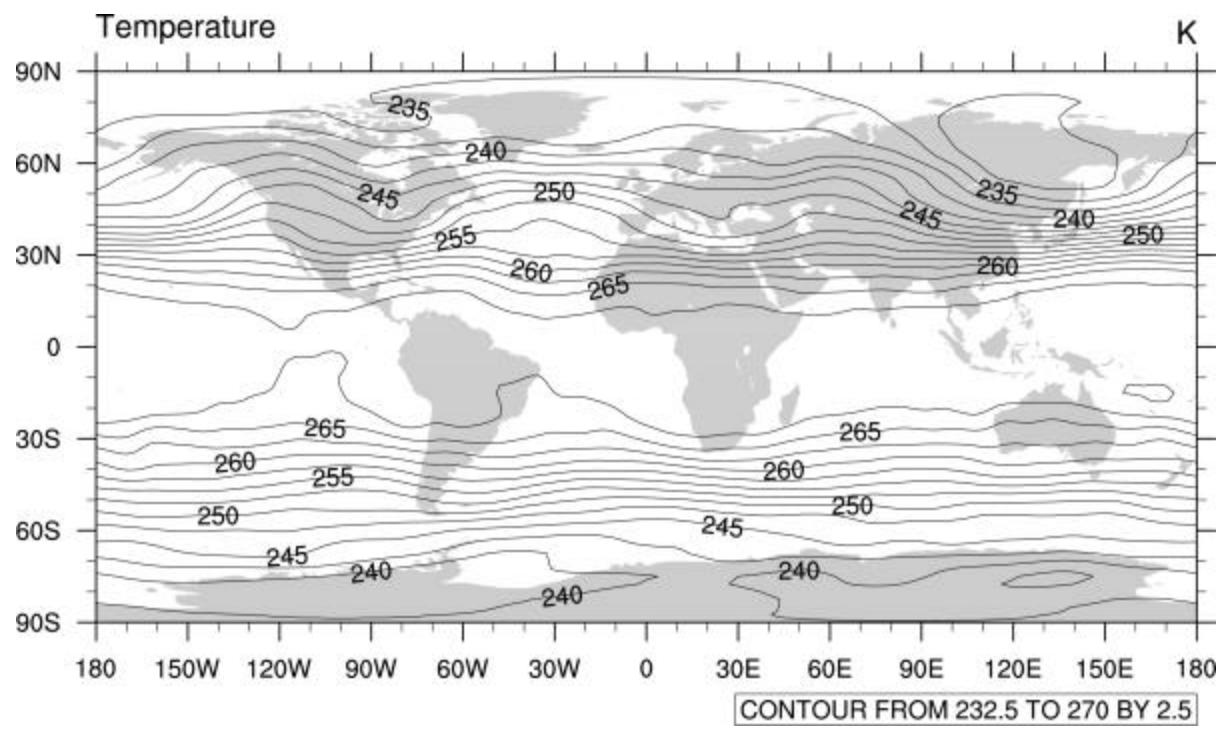
## Demo Script: Input/Function/Graphics

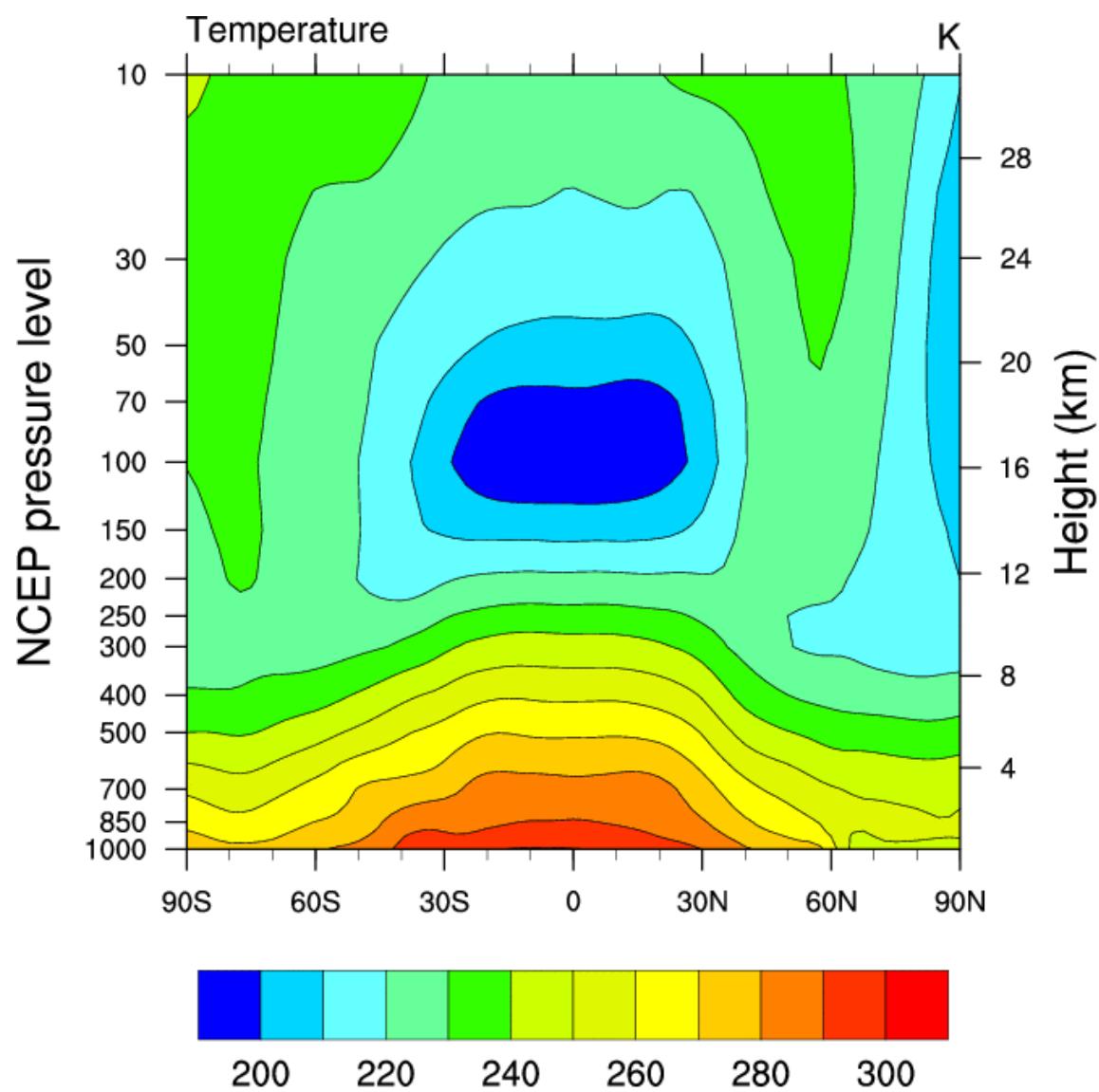
```
load "$NCARG_ROOT/lib/ncarg/nclscripts/csm/gsn_code.ncl"
load "$NCARG_ROOT/lib/ncarg/nclscripts/csm/gsn_csm.ncl"
load "$NCARG_ROOT/lib/ncarg/nclscripts/csm/contributed.ncl"
```

```
begin
  f = addfile("/ptmp/shear/TMP_1958-1997.nc", "r") ; open netCDF file
  t = f->T                                         ; input: t(time,lev,lat,lon)
                                                    ; (480,17,73,144)

  wks = gsn_open_wks("pdf", "demo")      ; open a graphic file
  gsn_define_colormap(wks,"gui_default") ; change from default
                                              ; cyl equi (standard/array subscripting)
  plot = gsn_csm_contour_map_ce(wks,t(0,5,:,:),False)
                                              ; polar stereo (standard+coordinate sub)
  plot = gsn_csm_contour_map_polar(wks,t(0,{500},:,:),False)
                                              ; pressure/hgt vs longitude {sub-sample}
  plot = gsn_csm_pres_hgt(wks, t(0,:,:{-20},{90:270:2}), False )

  res = True                                     ; change default plot
  res@cnFillOn = True                           ; use colors
  res@gsnSpreadColors = True                   ; use entire color map
  res@gsnMaximize = True                       ; max plot size
  plot = gsn_csm_pres_hgt(wks, t(0,:,:{180}), res) ; pass variable with plot options
end
```





# Demo Script: Array Syntax/Function/nc file

```
function brntV (t:numeric, ps[*][*][*]:numeric, hyam[*]:numeric, hybm[*]:numeric)
begin
p0    = 100000.                                ; Pa
R     = 277.04                                 ; J/(K*kg)
g     = 9.8                                    ; m/s
pHybrid = t                                     ; use built-in function
pHybrid = pres_hybrid_ccm (ps, p0, hyam, hybm) ; pHybrid(ntim,klev,nlat,mlon)
theta   = t                                     ; var-to-var, transfer meta data
theta   = t*(1000/pHybrid)^{0.286}              ; transfer meta data
                                                ; potential temperature [array op]

pReorder = pHybrid(time:,lat|:,lon|:,lev|:)    ; reorder dimensions
tReorder = theta(time:,lat|:,lon|:,lev|:)        ; reorder dimensions

brunt   = t                                     ; var-to-var, transfer meta data
brunt   = -g^2*pHybrid/(R*t*theta)*center_finite_diff(tReorder,pReorder,False,0)
return (brunt)
end
```

```
begin
f = addfile ("model.nc" , "r")                  ; open the file for reading ("r")
brunt = brntV( f->T , f->PS, f->hyam, f->hybm) ; invoke user function

g = addfile ("brunt.nc", "c")                   ; open nc file for creation ("c")
g->BRUNT = brunt
end
```

# Introduction to netCDF

NCL is based on a netCDF data model

# Fortran variable “data model”

**basic Fortran data object is an array**

- parameter (mlon=128, nlat=64)
- real x(mlon,nlat)

name: x  
type: real  
shape: 2D  
size: 128 by 64  
values: x(3,5) = 2.3

# netCDF variable data model

- **similar to basic fortran “data object”**
  - simple array oriented (dimensions)
  - name,type,size,shape,values (variables)

- **(possibly) has much more info: meta data**
  - information about the data
    - ☞ long\_name, units, etc
    - ☞ coordinate information
    - ☞ NCL can query, add and use meta data
    - ☞ meta data automatically used in some utilities:  
eg: gsn\_csm graphic functions

- **dimensions can be 1D variables**
  - x(lat,lon) lat/lon are 1D vectors (**coord variables**)

# netCDF files

- self describing
  - (ideally) all info contained within file
- portable

- netCDF file and variable limitations
  - 2GB limit (unless Large File Support [LFS] on system)
  - limited packing/compression options
  - only one “unlimited” dimension
  - limited data structures
    - ↳ no ragged arrays; no record/nested structures

- software
  - designed for generality, not high performance
  - no parallel writes; parallel reads are possible

# Examining a netCDF file

- `ncdump file_name | less`
  - dumps the entire contents of a file
- `ncdump -h file_name | less`
  - dumps the header info
  - NCL equivalent: `print (f)`
- `ncdump -v U file_name | less`
  - NCL equivalent: `print (U)`
- Note: `ncdump` is a Unidata utility
  - not a netCDF Operator (NCO)

# Parts of netCDF file

ncdump -h 1999.nc

## DIMENSIONS:

dimensions:

lat = 64 ;  
lon = 128 ;  
time = 12 ;

## FILE ATTRIBUTES:

global attributes:

:title = "Temp: 1999" ;  
:source = "NCAR" ;  
:conventions = "None" ;

## VARIABLES:

### Names , Attributes, Coordinates

variables:

```
float lat(lat) ;  
lat:long_name = "latitude" ;  
lat:units = "degrees_north" ;  
float lon(lon) ;  
lon:long_name = "longitude" ;  
lon:units = "degrees_east" ;  
int time(time) ;  
time:long_name = "time" ;  
time:units = "Month of Year" ;  
float T(time, lat, lon) ;  
T:long_name = "Temperature" ;  
T:units = "C" ;  
T:missing_value = 1.e+20f ;  
T:_FillValue = 1.e+20f
```

exercise: ncdump -h UV300.nc | less

# Detailed Look netCDF Variable (ncl)

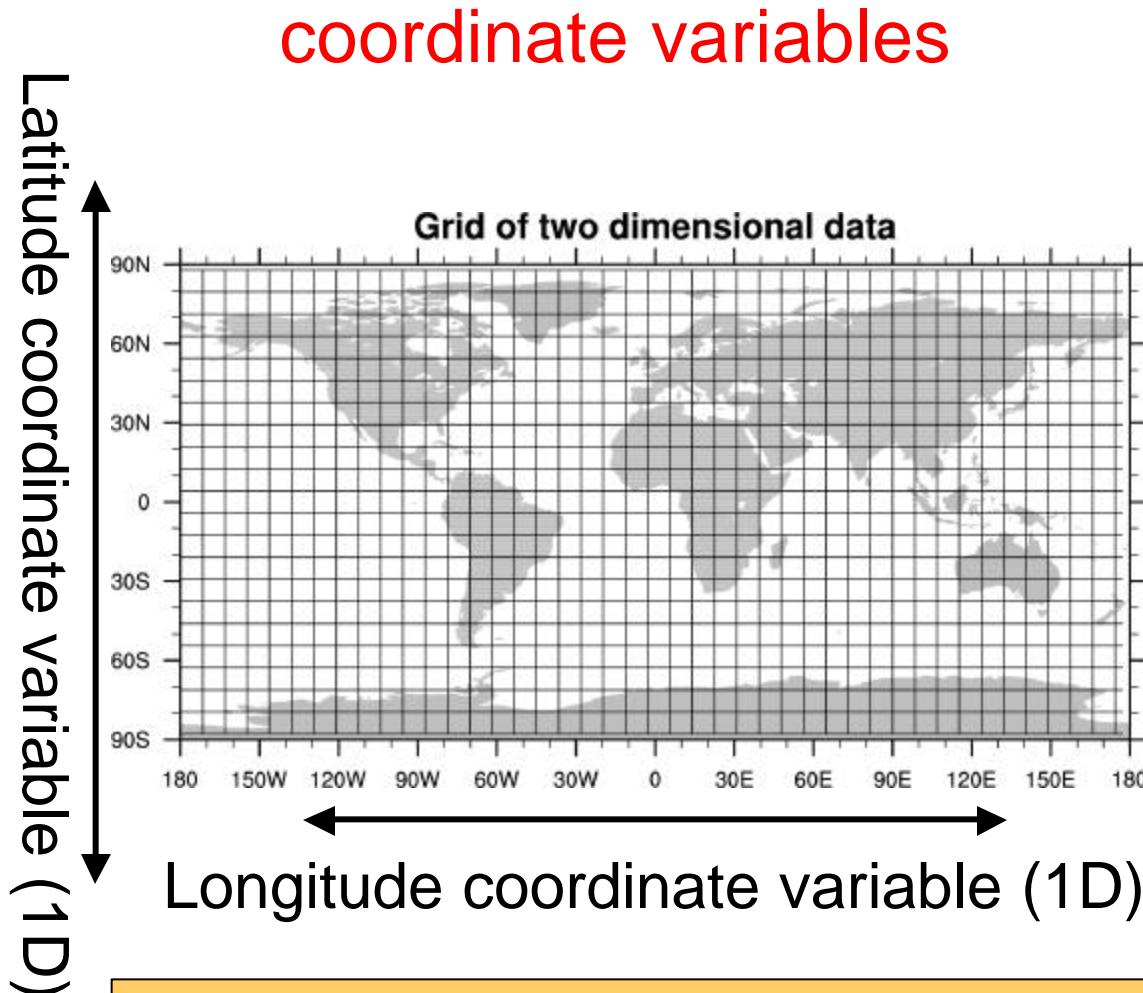
```
ncl <return> ; interactive mode  
ncl 0 > f = addfile ("UV300.nc", "r") ; open file  
ncl 1 > u = f->U ; import variable  
ncl 2 > printVarSummary (u) ; overview of variable
```

Variable: u  
Type: float  
Total Size: 65536 bytes  
16384 values  
Number of Dimensions: 3  
Dimensions and Sizes: [time | 2] x [lat | 64] x [lon | 128]  
Coordinates:  
    time: [ 1 .. 7 ]  
    lat: [ -87.8638 .. 87.8638 ]  
    lon: [ 0 .. 357.185 ]  
Number of Attributes: 5  
    \_FILLValue : 1e36  
    units : m/s  
    long\_name : Zonal Wind Component  
    short\_name : U  
    missing\_value : 1e36

Classic netCDF  
Variable Model

NCL  
syntax/funcs  
**query**  
**use**  
**modify**  
**add**  
any aspect of  
data object

# Picture: 2D netCDF Variable Model



attributes:

- long\_name
- units

NCL is **NOT LIMITED** to netCDF conforming variables

- eg: 2D coordinate arrays (curvilinear coords)

# UNLIMITED dimension

- **special dimension**
  - essentially a “record” dimension
  - time dimension is most frequently “unlimited”
  - used by the NCO to concatenate files
  - **no** special meaning to NCL
- **when creating output file in NCL (optional)**
  - **filedimdef** (outputfile, “time”, -1, True )
- **example:** ncdump - h T2m\_ud.nc

```
netcdf T2m_ud {  
dimensions:  
    time = UNLIMITED ; // (204 currently)  
    lat = 94 ;  
    lon = 192 ;  
    lev = 18}
```

**What is NCI?**

# Outline: Gross NCL Overview

- What NCL is
- Why learn NCL
- Setting up your environment
- Where to download the binaries
- Strengths and weaknesses
- How to run
- Useful URL's

# What NCL Is

- **Complete Programming Language**

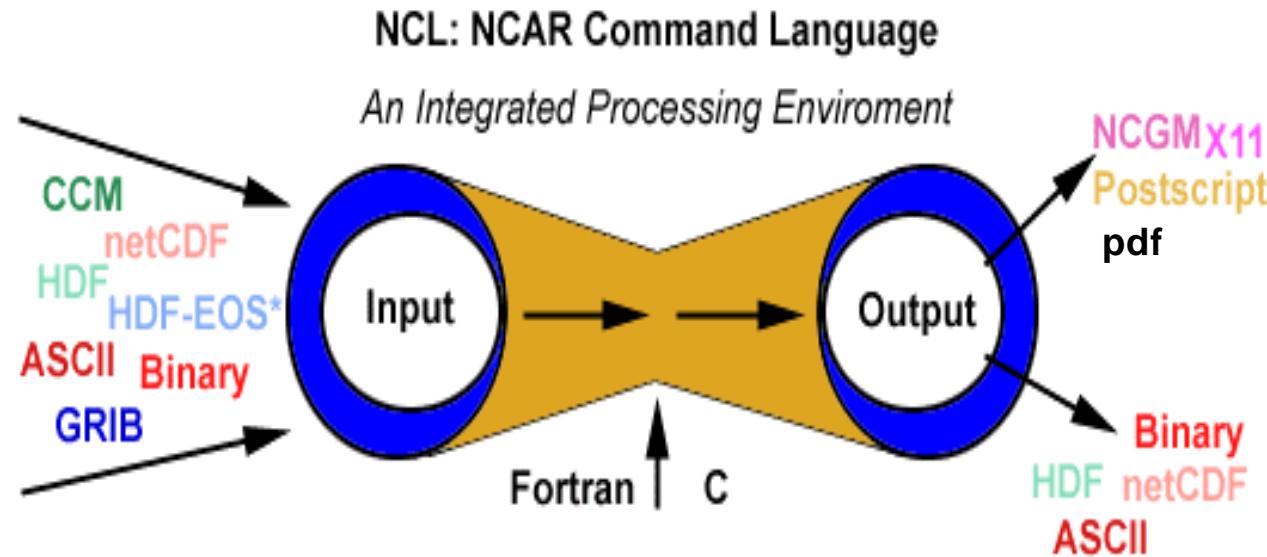
- data types
- variables
- operators
- expressions
- conditional statements
- loops
- functions/procedures

- **Extra features**

- query / manipulate meta data
- import data in a variety of formats
- array syntax [f90/95]
- can use user fortran/C codes and commercial libraries
- most functions/procedures ignore missing data

# Why Learn NCL

- **Integrated processing environment**



- **freeware: supported public domain software**
- **portable: Linux/Unix, Windows, MacOSX**
- **general purpose: many unique capabilities**
- **excellent graphics**

# Paths and Software

- **NCL runs on:**

- most (?all?) UNIX/linux systems
- Windows NT [need GNU Cygwin and XFree86]
- MAC OS X

- **Binaries ... Free**

- <http://ngwww.ucar.edu/ncl/download.htm>
- Mary Haley [haley@ucar.edu](mailto:haley@ucar.edu) is the POC

- **NCAR PATH** dataproc/dave/blackforest/CGD

- **/contrib/bin/ncl**

- ☛ setenv NCARG\_ROOT /contrib

- ☛ setenv PATH \${NCARG\_ROOT}/bin:\${PATH}

- **Non-NCAR:** contact your system admin

# Useful NCL related URLs

- **CGD/CSM and SCD links**

- <http://www.cgd.ucar.edu/csm/support/>
- <http://ngwww.ucar.edu/ncl/>

- **NCL functions and procedures (built-in)**

- <http://ngwww.ucar.edu/ngdoc/ng/ref/ncl/NclFuncAndProcRef.html>

- **Overview of NCL Language**

- <http://ngwww.ucar.edu/ngdoc/ng /ref/ncl/Overview.html>
- <http://www.cgd.ucar.edu/csm/support/Document/manual.shtml>

- **GSUN: Getting Started Using NCL**

- <http://ngwww.ucar.edu/ngdoc/ng/ncl/gsun/>
- <http://ngwww.ucar.edu/ngdoc/ng/ncl/gsun/appendixd.html>

- **Listing of NCL terms**

- <http://ngwww.ucar.edu/ngdoc/ng/nggenrl/ngindexALL.html>

# Strengths/Weaknesses

- **Strengths**

- file handling (netCDF, HDF, HDF-EOS, GRIB)
- many array processing functions
- automatically handles missing values [ `_FillValue` ]
- excellent graphics
- supported by NCAR SCD and CGD
- can use f77/f90 subroutines , C/C++, commercial libs

- **Weaknesses**

- limited character IO (better to invoke a F/C routine)
- does not support complex numbers
- documentation: lots of it BUT fragmented
- no command line arguments (**way around this**)

- **Nuisance**

- no “else if” ; “end if”, “end do” [remember the space!!]
- (most) functions do not copy meta data: **contributed.ncl**

# NCL

- interpreted lang {idl,matlab}
  - no optimization
- case sensitive
- subscripts are 0 based
- row major
- right index fastest varying
- \ continue next line
- ; comment
- pass by reference
- automatic memory
- + string concatenation
- built in file handling for nc, grb, ccm, hdf, (hdfeos )

# Fortran {C}

- compiled language
  - high optimization {good}
- case insensitive {case senitive}
- subscripts are 1 based {0}
- column major {row}
- left index fastest varying {right}
- col 6 [f77]; ; & [f90] continue
- C [f77]; ! [f90] comment {\\* ... \*/}
- pass by reference {value}
- manual memory allocation
- // char concatenation {strcat}

# Running NCL

- **Interactive Mode (Command line)**

prompt> **ncl** <return>

ncl> enter commands (**no** begin/end)

ncl> **quit** <return>

- can save interactive commands

ncl> **record** ("file\_name")

ncl> **stop record**

- **Batch Mode** [**<** and **.ncl** are optional]

- prompt> **ncl** < script.ncl

- **ncl** script.ncl [also acceptable]

- prompt> **ncl** < script.ncl **>!** script.out

- prompt> **ncl** < script.ncl **>!** script.out **&**

- ☞ appending "**&**" means put in background

- ☞ note: the **>!** **&** are appropriate for csh and tcsh

# Language Basics

# Outline: Language Basics

- data types
- variables
- attributes
- `_FillValue`
- named dimensions
- coordinate variables
- **print** and **printVarSummary**
- shaping
- subscripting

# Data Types

## Numeric

- double (64 bit)
- float (32 bit)
- long (32 bit) [SGI 64]
- integer (32 bit)
- short (16 bit)
- byte ( 8 bit, 0-255)
- complex NOT supported

## non-Numeric

- string
- character
- graphic
- file
- logical
- list

# Conversion between data types

- **coersion**
  - implicit conversion of one type to another
- **automatic coersion when no info is lost**
  - fortran:  $x=i$  and  $i=x$
  - NCL:  $x=i$  and  $i=\text{floattointeger}(x)$
- **many functions to perform conversions**
  - <http://ngwww.ucar.edu/ngdoc/ng/ref/ncl/NclFuncAndProcRef.html>

# Variables based on netCDF model

- **Two types (differ only in how referenced)**

- regular variables: exist in memory [like F/C]
- file variables: exist in a file

- **Numerous functions can query variables**

|                                      |                                 |
|--------------------------------------|---------------------------------|
| - <a href="#">isfilevar</a>          | <a href="#">isvar</a>           |
| - <a href="#">isfilevaratt</a>       | <a href="#">isatt</a>           |
| - <a href="#">getfilevaratts</a>     | <b>getvaratts</b>               |
| - <a href="#">getfilevardimsizes</a> | <b>dimsizes</b>                 |
| - <a href="#">getfilevardims</a>     | <a href="#">getvardims</a>      |
| - <b>getfilevarnames</b>             | <a href="#">getfilevartypes</a> |

<http://ngwww.ucar.edu/ngdoc/ng/ref/ncl/NclFuncAndProcRef.html#FileFunctions>

<http://ngwww.ucar.edu/ngdoc/ng/ref/ncl/NclFuncAndProcRef.html#NCLVariableFunctions>

# Variable Assignment

```
a_int = 1      , a_float = 2.0 [2e-5] , a_double = 3.2d0  
a_string = "a" , a_logical = True
```

- **Variable-to-Variable assignment**

- consider  $y = x$  where  $x$  is previously defined
  - ☞ if  $y$  not defined:
    - ☞  $y$  has same type/shape/values/ meta data as  $x$
  - ☞ if  $y$  predefined:
    - ☞  $y$  must have same shape and type
    - ☞ or “ $x$ ” must be **coerceible** to the type of  $y$
    - ☞  $y$  attributes, dimension names and coordinate variables, if they exist, will be over written

- **Value-only assignment (no meta copy)**

- $U$  multi-dimensional array with meta data
  - ☞  $Uval = ( / U / )$  or  $Uval = ( / f->U / )$
  - ☞ the  $( / \dots / )$  operator pair strips meta data

# Attributes

- **info about a variable or file**
  - attributes can be any data type but **file** or **list**
  - scalar, multi dimensional array (string, numeric)

- **assign/access with @ character**
  - `T@long_name = "temperature"`
  - `T@wgts = (/ 0.25, 0.5, 0.25 /)`
  - `T@x3d = x3D`
  - `T @_FillValue = -999.`
  - `title = x@long_name`

- **attribute functions** [`isatt`, `getfilevaratts`]
  - if (`isatt(T,"units")`) then .... end if
  - `atts = getfilevaratts (fin, "T")`
- **delete** can eliminate an attribute
  - `delete(T@long_name)`

## \_FillValue

- **Unidata and NCL reserved attribute**
- **most NCL functions ignore \_FillValue**
  - “missing\_value” has **no** special status to NCL
  - if “T” has “missing\_value” but no “\_FillValue”
    - ☞  $T@_FillValue = T@missing_value$
- **NCO looks for “missing\_value”**
  - best to create netCDF files with both
- **NCL: best to not use zero as a \_FillValue**
  - OK except when contouring [random bug]

# Dimensions

- **May be “named”**
  - provides alternative way to reference subscripts
  - recommendation: always name dimensions

- **Assigned with ! Character** {let  $T(:,:,,:)$ }
  - $T!0 = "time"$  ; leftmost [slowest varying] dim
  - $T!1 = "lat"$
  - $T!2 = "lon"$  ; rightmost [fastest varying] dim

- **Dim names may be renamed, retrieved**
  - $T!1 = "LAT" \dots dName = T!2$
- **delete** can eliminate:  $\text{delete}(T!2)$

- **Named dimensions used to reshape**

# Coordinate Variables

- **data coordinates (strict netCDF definition)**
  - 1D array
  - monotonically in[de]creasing numeric values
  - can only be assigned to a named dimension
  - 1D array must be the same size as dimension
  - assigned via & character
    - ☞  $T\&lat = \text{latitude}$

- **used in graphics, coordinate subscripting**

- **coordinate functions**
  - `iscoord` , `isfilevarcoord`
- **delete** can eliminate coordinate array
  - `delete(T&time)`

# Create and Assign Coordinate Variables

- **create 1D vector array**

- time = (/ 1980, 1981, 1982 /) ; integer
- lon = ispan(0, 355, 5)\*1.0 ; integer->float

- **assign dimension name(s)**

- time!0 = “time”
- lon!0 = “lon”

- **assign vector array to itself**

- time&time = time
- lon&lon = lon

- **let x be 2D: name dimensions**

- x!0 = “time” ... x!1 = “lon”

- **assign coordinate variables to x**

- x&time = time ... x&lon = lon

# Access/Change/Create/Delete Meta Data

- **@ attributes**

- u@long\_name = "U"
  - lonName = u@long\_name

- **! named dimensions**

- u!0 = "TIME"
  - tName = u!0

- **& coordinate variable**

- u&lat = (/ -90., -85, .... , 85., 90. /)
  - latitude = u&lat

- **\$ substitute string**

- x = fin->\$variable(n)\$ ... x = fin->\$\$Temp\$\$

# printVarSummary

- Print out variable (data object) information
  - type
  - dimension information
  - coordinate information (if present)
  - attributes (if present)
- **printVarSummary (u)**

Variable: u

Type: double

Total Size: 1179648 bytes  
147456 values

Number of Dimensions: 4

Dimensions / Sizes: [time | 1] x [lev | 18] x [lat | 64] x [lon | 128]

Coordinates:

time: [4046..4046]

lev: [4.809 .. 992.5282]

lat: [-87.86379 .. 87.86379]

lon: [ 0. 0 .. 357.1875]

Number of Attributes: 2

long\_name: zonal wind component

units: m/s

# print (1 of 3)

- Prints out all variable information including
  - meta data, values
  - T(lat,lon): print (T)

Variable: T

Type: float

Total Size: 32768 bytes

8192 values

Number of Dimensions: 2

Dimensions / Sizes: [lat | 64] x [lon | 128]

Coordinates:

lat: [-87.86379 .. 87.86379]

lon: [ 0. 0 .. 357.1875]

Number of Attributes: 2

long\_name: Temperature

units: C

(0,0) -31.7

(0,1) -31.4

(0,2) -32.3

(0,3) -33.4

(0,4) -31.3 etc. [entire T array will be printed]

## print (2 of 3)

- can be used to print a subset of array
  - meta data, values
  - T(lat,lon): `print( T(:,103) )` or `print( T(:,{110}) )`

Variable: T (subsection)

Type: float

Total Size: 256 bytes

64 values

Number of Dimensions: 1

Dimensions / Sizes: [lat | 64]

Coordinates:

lat: [-87.86379 .. 87.86379]

Number of Attributes: 3

long\_name: Temperature

units: C

Ion: 109.6875 [ added ]

(0) -40.7

(1) -33.0

(2) -25.1

(3) -20.0

(4) -15.3 etc.

## print (3 of 3)

- **print with embedded strings**

- no meta data

- `print ( "min(T)="+min(T)+" max(T)="+max(T) )`

```
(0) min(T)=-53.8125 max(T)=25.9736
```

- **sprintf and sprinti provide formatting**

- often used in graphics

- `print ( "min(T) = "+ sprintf("%5.2f ", min(T)) )`

```
(0) min(T) = -53.81
```

- **sprinti can left fill with zeros (ex: let n=3)**

- `fnam = "h" + sprinti ("%0.5i", n) + ".nc"`

- `print("file name = "+fnam)`

```
(0) file name = h00003.nc
```

# **write\_matrix(x[\*][\*], fmt, opt)**

- **pretty-print 2D array to standard out**
  - integer, float, double
  - user format control (fmt)
  - T(5,7): **write\_matrix (T, “5f7.2”, False)**

|       |       |       |       |       |
|-------|-------|-------|-------|-------|
| 4.35  | 4.39  | 0.27  | -3.35 | -6.90 |
| 4.36  | 4.66  | 3.77  | -1.66 | 4.06  |
| 9.73  | -5.84 | 0.89  | 8.46  | 10.39 |
| 4.91  | 4.59  | -3.09 | 7.55  | 4.56  |
| .17   | 3.68  | 5.08  | 0.14  | -5.63 |
| -0.63 | -4.12 | -2.51 | 1.76  | -1.43 |
| -4.29 | 0.07  | 5.85  | 0.87  | 8.65  |

- **can also create an ASCII file**

```
opt      = True
opt@fout = "foo.ascii" ; file name
write_matrix (T, "5f7.2", opt)
```

# Variable Shaping

- transpose, thin, reorder dimensions
- functions **may** require data in specific order
  - map plot functions want array order  $T(\dots, \text{lat}, \text{lon})$
- **can and should be done without loops**
  - use NCL syntax or functions
  - very fast for variables in memory
- how? ... two approaches: let  $T(\text{time}, \text{lat}, \text{lon})$ 
  - named dimensions:  $t = T(\text{lat}[:, \text{lon}[:, \text{time}[]])$
  - NCL functions:
    -  **ndtooned**:  $t1D = \text{ndtooned}(T)$
    -  **onedtond**:  $t2D = \text{onedtond}(t1D, (/N,M/))$

# Variable Subscripting (1 of 3)

## Standard Array Subscripting [similar to f90]

- ranges: start/end and [optional] stride
- indices separated by `:`
- omitting start/end index
  - implies starting at begin or end

Consider  $T(\text{time}, \text{lat}, \text{lon})$

|                  |                                                                                     |                                                                                     |
|------------------|-------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------|
| $T$              |   | entire array [ don't use $T(:,:, :)$ ]                                              |
| $T(0,:,:5)$      |  | 1 <sup>st</sup> time index, all lat, every 5 <sup>th</sup> lon                      |
| $T(0,:,:-1,:50)$ |  | 1 <sup>st</sup> time index, reverse lat order, 1 <sup>st</sup> 51 lon               |
| $T(:1,45,10:20)$ |  | 1 <sup>st</sup> 2 time indices, 46 <sup>th</sup> index of lat, 10-20 indices of lon |

# Variable Subscripting 2 of 3)

## Coordinate Variable Subscripting

- **only** applies to netCDF conforming variables
- same rules apply for ranges, strides, defaults
- use curly brackets **{...}**
- standard and coordinate subs can be mixed  
[if no reorder]

T(:,{-30:30},:)



all times/Ion, lat -30° to +30°  
(inclusive)

T(0,{-20},{-180:35:3})



1<sup>st</sup> time, lat nearest - 20°,  
every 3rd Ion between -180°  
and 35°

# Variable Subscripting 3 of 3)

## Named Dimensions

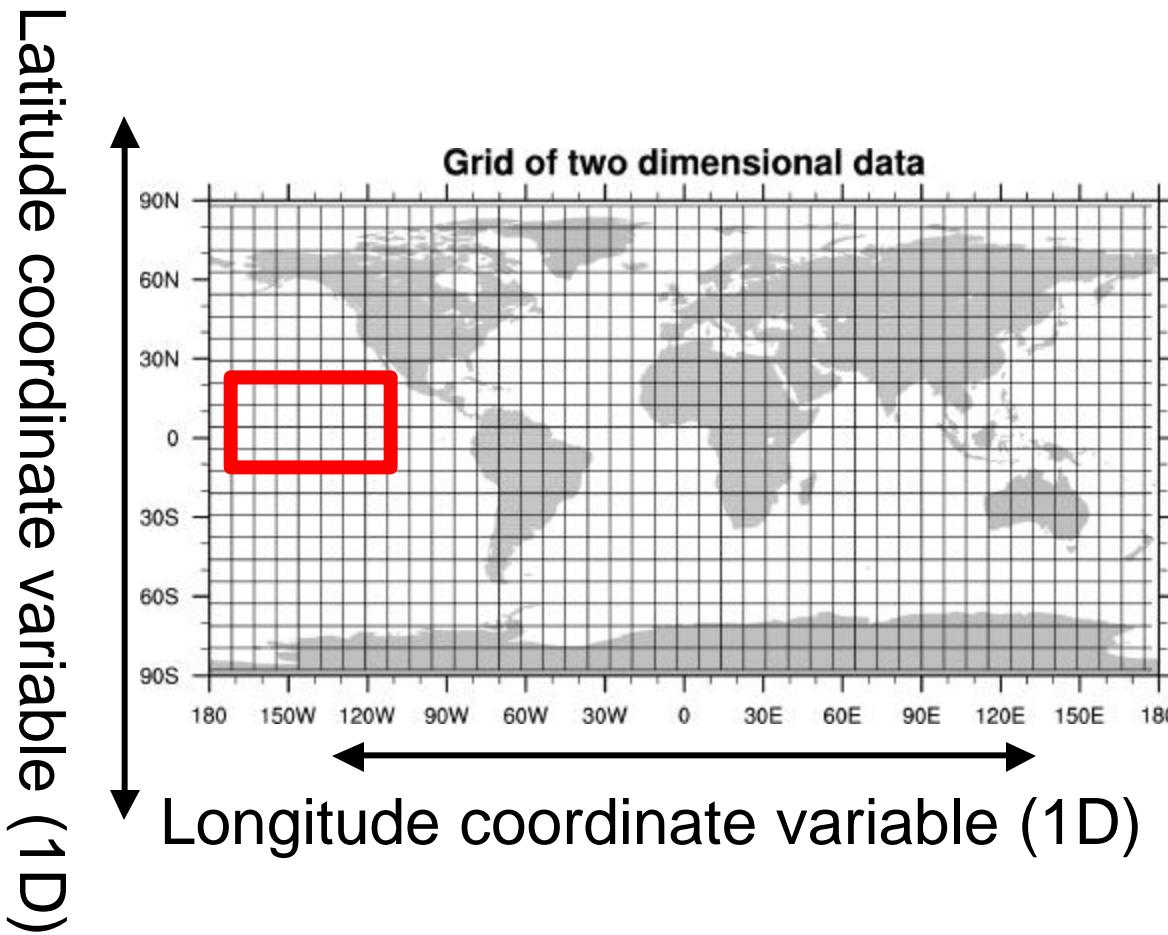
- only used for dimension reordering
- indicated by |
- dimension names must be used for each subscript
- named/coordinate subscripting can be mixed

Consider  $T(\text{time}, \text{lat}, \text{lon})$

$t = T(\text{lat}|:|, \text{lon}|:|, \text{time}|:|)$  ↗ makes  $t(\text{lat}, \text{lon}, \text{time})$

$t = T(\text{time}|:|, \{\text{lon}|90:120\}, \{\text{lat}|{-}20:20\})$  ↗ all times,  
90-120° lon, -20-20° lat

# Standard and Coordinate Subscripting



Standard:

$T(9:13,1:8)$

Coordinate:

$T(\{-10:20\}, \{-170:-110\})$

Combined:

$T(\{-10:20\}, 1:8)$